Revisiting Natural Gradient for Deep Networks

Razvan Pascanu and Yoshua Bengio
Dept. IRO
University of Montreal
Montreal, QC

Abstract

The aim of this paper is three-fold. First we show that Hessian-Free (Martens, 2010) and Krylov Subspace Descent (Vinyals and Povey, 2012) can be described as implementations of natural gradient descent due to their use of the extended Gauss-Newton approximation of the Hessian. Secondly we re-derive natural gradient from basic principles, contrasting the difference between two versions of the algorithm found in the neural network literature, as well as highlighting a few differences between natural gradient and typical second order methods. Lastly we show empirically that natural gradient can be robust to overfitting and particularly it can be robust to the order in which the training data is presented to the model.

1 Introduction

Several recent papers tried to address the issue of using better optimization techniques for machine learning, especially for training deep architectures or neural networks of various kinds. Hessian-Free optimization (Martens, 2010; Sutskever *et al.*, 2011; Chapelle and Erhan, 2011), Krylov Subspace Descent (Vinyals and Povey, 2012), natural gradient descent (Amari, 1997; Park *et al.*, 2000; Le Roux *et al.*, 2008; Le Roux *et al.*, 2011) are just a few of such recently proposed algorithms. They usually can be split in two different categories: those which make use of second order information and those which use the geometry of the underlying parameter manifold (natural gradient).

One particularly interesting pipeline to scale up such algorithms was originally proposed in Pearlmutter (1994), finetuned in Schraudolph (2001) and represents the backbone behind both Hessian-Free optimization (Martens, 2010) and Krylov Subspace Descent (Vinyals and Povey, 2012). The core idea behind it is to make use of the forward (renamed to *R-operator* in Pearlmutter (1994)) and backward pass of automatic differentiation to compute efficient products between Jacobian or Hessian matrices and vectors. These products are used within a truncated-Newton approach (Nocedal and Wright, 2000) which considers the exact Hessian and only inverts it approximately without the need for explicitly storing the matrix in memory, as opposed to other approaches which perform a more crude approximation of the Hessian (or Fisher) matrix (either diagonal or block-diagonal).

The contributions of this paper to the study of the natural gradient are as follows. We provide a detailed derivation of the natural gradient, avoiding elements of information geometry. We distinguish natural gradient descent from TONGA and provide arguments suggesting that natural gradient may also benefits from a form of robustness that should yield better generalization. The arguments for this robustness are different from those invoked for TONGA. We show experimentally the effects of this robustness when we increase the accuracy of the metric using extra *unlabeled data*. We also provide evidence that the natural gradient is robust to the order of training examples, resulting in lower variance as we change the order. The final contribution of the paper is to show that Martens' Hessian-Free approach of Martens (2010) (and implicitly Krylov Subspace Descent (KSD) algorithm) can be cast into the framework of the natural gradient, showing how these methods can be seen as doing natural gradient rather then second order optimization.

2 Natural Gradient

Natural gradient can be traced back to Amari's work on information geometry (Amari, 1985) and its application to various neural networks (Amari *et al.*, 1992; Amari, 1997), though a more in depth introduction can be found in Amari (1998); Park *et al.* (2000); Arnold *et al.* (2011). The algorithm has also been successfully applied in the reinforcement learning community (Kakade, 2001; Peters and Schaal, 2008) and for stochastic search (Sun *et al.*, 2009). Le Roux *et al.* (2007) introduces a different formulation of the algorithm for deep models. Although similar in name, the algorithm is motivated differently and is not equivalent to Amari's version, as will be shown in section 4.1.

Let us consider a family of density functions $\mathcal{F} : \mathbb{R}^P \rightarrow (\mathbb{B} \rightarrow [0, 1])$, where for every $\theta \in \mathbb{R}^P$, $\mathcal{F}(\theta)$ defines a density function from $\mathbb{B} \rightarrow [0, 1]$ over the random variable $\mathbf{z} \in \mathbb{B}$, where $\mathbb{B}$ is some suitable numeric set of values, for e.g. $\mathbb{B} = \mathbb{R}^N$. We also define a loss function that we want to minimize $\mathcal{L} : \mathbb{R}^P \rightarrow \mathbb{R}$. Any choice of $\theta \in \mathbb{R}^P$ defines a particular density function $p_\theta(\mathbf{z}) = \mathcal{F}(\theta)$ and by considering all possible θ values, we explore the set $\mathcal{F}$, which is our functional manifold. Because we can define a similarity measures between nearby density functions, given by the KL-divergence which in its infinitesimal form behaves like a distance measure, we are dealing with a Riemannian manifold whose metric is given by the Fisher Information matrix. Natural gradient attempts to move along the manifold by correcting the gradient of $\mathcal{L}$ according to the local curvature of the KL-divergence surface [1]:

$$\nabla_N \mathcal{L}(\theta) = \frac{\partial \mathcal{L}(\theta)}{\partial \theta} \mathbb{E}_{\mathbf{z}} \left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta} \right)^T \left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta} \right) \right]^{-1} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta} \mathbf{G}^{-1} \tag{1}$$

We can derive this result *without relying on information geometry*. We consider the natural gradient to be defined as the algorithm which, at each step, picks a descent direction such that the KL-divergence between p_θ and $p_{\theta+\Delta\theta}$ is constant. At each step, we need to find $\Delta\theta$ such that:

$$\begin{array}{l} \arg\min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta) \\ \text{s. t. } KL(p_\theta || p_{\theta+\Delta\theta}) = constant \end{array} \tag{2}$$

Using this constraint we ensure that we move along the functional manifold with constant speed, without being slowed down by its curvature. This also makes learning robust to re-parametrizations of the model, as the functional behaviour of p does not depend on how it is parametrized.

Assuming $\Delta\theta \rightarrow 0$, we can approximate the KL divergence by its Taylor series:

$$\begin{aligned} KL(p_\theta(\mathbf{z}) \parallel p_{\theta+\Delta\theta}(\mathbf{z})) &\approx (\mathbb{E}_{\mathbf{z}}[\log p_\theta] - \mathbb{E}_{\mathbf{z}}[\log p_\theta]) - \mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta}{\partial \theta}\right] \Delta\theta - \frac{1}{2} \Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[\frac{\partial^2 \log p_\theta}{\partial \theta^2}\right] \Delta\theta \\ &= \frac{1}{2} \Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta(\mathbf{z})}{\partial \theta^2}\right] \Delta\theta \qquad (3) \\ &= \frac{1}{2} \Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T \left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right] \Delta\theta \qquad (4) \end{aligned}$$

The first term cancels out and because $\mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right] = 0$, [2] we are left with only the last term. The Fisher Information Matrix form can be obtain from the expected value of the Hessian through algebraic manipulations (see the Appendix).

We now express equation (2) as a Lagrangian, where the KL divergence is approximated by (4) and $\mathcal{L}(\theta + \Delta\theta)$ by its first order Taylor series $\mathcal{L}(\theta) + \frac{\partial \mathcal{L}(\theta)}{\partial \theta} \Delta\theta$:

[1] Throughout this paper we use the mathematical convention that a partial derivative $\frac{\partial \log p_\theta}{\partial \theta}$ is a row-vector

[2] Proof: $\mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right] = \sum_{\mathbf{z}} \left(p_\theta(\mathbf{z}) \frac{1}{p_\theta(\mathbf{z})} \frac{\partial p_\theta(\mathbf{z})}{\partial \theta} \right) = \frac{\partial}{\partial \theta} \left(\sum_\theta p_\theta(\mathbf{z}) \right) = \frac{\partial 1}{\partial \theta} = 0$. The proof holds for the continuous case as well, replacing sums for integrals.

$$\mathcal{L}(\theta) + \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\Delta\theta + \frac{1}{2}\lambda \Delta\theta^T \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T \left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right]\Delta\theta = 0 \tag{5}$$

Solving equation (5) for $\Delta\theta$ gives us the natural gradient formula (1). Note that we get a scalar factor of $2\frac{1}{\lambda}$ times the natural gradient. We fold this scalar into the learning rate, and hence the learning rate also controls the difference between p_θ and $p_{\theta+\Delta\theta}$ that we impose at each step. Also the approximations we make are meaningful only around θ. Schaul (2012) suggests that using a large step size might be harmful for convergence. We deal with such issues both by using damping (i.e. setting a trust region around θ) and by properly selecting a learning rate.

3 Natural Gradient for Neural Networks

The natural gradient for neural networks relies on their probabilistic interpretation (which induces a similarity measure between different parametrization of the model) given in the form of conditional probabilities $p_\theta(\mathbf{t}|\mathbf{x})$, with $\mathbf{x}$ representing the input and $\mathbf{t}$ the target.

We make use of the following notation. $q(\mathbf{x})$ describes the data generating distribution of $\mathbf{x}$ and $q(\mathbf{t}|\mathbf{x})$ is the distribution we want to learn. $\mathbf{y}$ is the output of the model, and by an abuse of notation, it will refer to either the function mapping inputs to outputs, or the vector of output activations. $\mathbf{r}$ is the output of the model before applying the output activation function σ. $\mathbf{t}^{(i)}$ and $\mathbf{x}^{(i)}$ are the i-th target and input samples of the training set. $\mathbf{J}_\mathbf{y}$ stands for the Jacobian matrix $\mathbf{J}_y = \begin{bmatrix} \frac{\partial y_1}{\partial \theta_1} & .. & \frac{\partial y_1}{\partial \theta_P} \\ .. & .. & .. \\ \frac{\partial y_o}{\partial \theta_1} & .. & \frac{\partial y_0}{\partial \theta_P} \end{bmatrix}$. Finally, lower indices such as y_i denote the i-th element of a vector.

We define the neural network loss as follows:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_i^n \left[\log p_\theta(\mathbf{t}^{(i)}|\mathbf{y}(\mathbf{x}^{(i)}))\right] = \frac{1}{n}\sum_i^n \left[\log p_\theta(\mathbf{t}^{(i)}|\sigma(\mathbf{r}(\mathbf{x}^{(i)})))\right] \tag{6}$$

Because we have a conditional density function $p_\theta(\mathbf{t}|\mathbf{x})$ the formulation for the natural gradient changes slightly. Each value of $\mathbf{x}$ now defines a different family of density functions $p_\theta(\mathbf{t}|\mathbf{x})$, and hence a different manifold. In order to measure the functional behaviour of $p_\theta(\mathbf{t}|\mathbf{x})$ for different values of $\mathbf{x}$, we use the expected value (with respect to $\mathbf{x} \sim \tilde{q}(\mathbf{x})$) of the KL-divergence between $p_\theta(\mathbf{t}|\mathbf{x})$ and $p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x})$.

$$\begin{aligned} &\arg\min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta) \\ &\text{s. t. } \mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[KL(p_\theta(\mathbf{t}|\mathbf{x})||p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x}))\right] = constant \end{aligned} \tag{7}$$

The metric $\mathbf{G}$ is now an expectation over $\tilde{q}(\mathbf{x})$ of an expectation over $p(\mathbf{t}|\mathbf{x})$. The former averages over possible manifolds generated by different choices of $\mathbf{x}$, while the latter comes from the definition of the Fisher Information Matrix.

$$\nabla_N \mathcal{L}(\theta) = \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[\mathbb{E}_{\mathbf{t}\sim p(\mathbf{t}|\mathbf{x})}\left[\left(\frac{\partial \log p_\theta(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)^T \left(\frac{\partial \log p_\theta(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)\right]\right]^{-1} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\mathbf{G}^{-1} \tag{8}$$

Note that we use the distribution $\tilde{q}$ instead of the empirical q. This is done in order to emphesis that the theory does not force us to use the empirical distribution. However, in practice, we do want $\tilde{q}$ to be as close as possible to q such that the curvature of the KL-divergence matches (in some sense) the curvature of the error surface. To clarify the effects of $\tilde{q}$ let us consider an example. Assume that $\tilde{q}$ is unbalanced with respect to q. Namely it contains twice the amount of elements of a class $\mathbf{A}$ versus the other $\mathbf{B}$. This means that a change in θ that affects elements of class $\mathbf{A}$ is seen as having a larger impact on p (in the KL sense) than a change that affects the prediction of elements in $\mathbf{B}$. Due to the formulation of natural gradient, we will move slower along any direction that affects $\mathbf{A}$ at the expense of $\mathbf{B}$ landing with higher probability on solutions θ^* that favour predicting $\mathbf{A}$. In practice we approximate this expectation over $\tilde{q}$ by a sample average over minibatches.

In what follows we consider typical output activation functions and the metrics $\mathbf{G}$ they induce. In the Appendix we provide a detailed description of how these matrices were obtained starting from equation (8). Similar derivations were done in Park *et al.* (2000), which we repeat for convenience. The formulas we get for the linear, sigmoid and softmax activation functions are:

$$\mathbf{G}_{linear} = \beta^2 \mathbb{E}_{\mathbf{x}\sim\tilde{q}} \left[\frac{\partial \mathbf{y}}{\partial \theta}^T \frac{\partial \mathbf{y}}{\partial \theta} \right] = \beta^2 \mathbb{E}_{\mathbf{x}\sim\tilde{q}} \left[\mathbf{J}_{\mathbf{y}}^T \mathbf{J}_{\mathbf{y}} \right] \tag{9}$$

$$\mathbf{G}_{sigmoid} = \mathbb{E}_{\mathbf{x}\sim\tilde{q}} \left[\mathbf{J}_{\mathbf{y}}^T \text{diag}(\frac{1}{\mathbf{y}(1-\mathbf{y})}) \mathbf{J}_{\mathbf{y}} \right] \tag{10}$$

$$\mathbf{G}_{softmax} = \mathbf{E}_{\mathbf{x}\sim\tilde{q}} \left[\sum_i^o \frac{1}{\mathbf{y}_i} \left(\frac{\partial y_i}{\partial \theta} \right)^T \frac{\partial y_i}{\partial \theta} \right] \tag{11}$$

To efficiently implement the natural gradient, we use a truncated Newton approach following the same pipeline as Hessian-Free (Martens, 2010) (more details are provided in the Appendix). We rely on Theano (Bergstra *et al.*, 2010) for both flexibility and in order to use GPUs to speed up. The advantages of this pipeline are two-fold: (1) it uses the full-rank matrix, without the need for explicitely storing it in memory and (2) it *does not rely on a smoothness assumption* of the metric. Unlike other algorithms such as nonlinear conjugate gradient or BFGS, it does *not* assume that the curvature changes slowly as we change θ. This seems to be important for recurrent neural networks (as well as probably for deep models) where the curvature can change quickly (Pascanu *et al.*, 2013).

4 Insights into natural gradient

Figure 1 considers a one hidden unit auto-encoder, where we minimize the error $(x - w \cdot \text{sigmoid}(wx + b) + b)^2$ and shows the path taken by Newton method (blue), natural gradient (gray), Le Roux's version of natural gradient (orange) and gradient descent (purple). On the left (larger plot) we show the error surface as a contour plot, where the x-axis represents b and y-axis is w. We consider two different starting points ([1] and [3]) and draw the first 100 steps taken by each algorithm towards a local minima. The length of every other step is depicted by a different shade of color. Hyper-parameters, like learning rate and damping constant, were chosen such to improve convergence speed while maintaining stability (i.e. we looked for a smooth path). Values are provided in the Appendix. At every point θ during optimization, natural gradient considers a different KL divergence surface, $KL(p_\theta || p_{\theta+\Delta\theta})$ parametrized by $\Delta\theta$, which has a minima at origin. On the right we have contour plots of four different KL surfaces. They correspond to locations indicated by black arrows on the path of natural gradient. The x-axis is Δb and y-axis is Δw for the KL surfaces subplots. On top of these contour plots we show the direction and length of the steps proposed by each of the four considered algorithms.

The point of this plot is to illustrate that each algorithm can take a different path in the parameter space towards local minima. In a regime where we have a non-convex problem, with limited resources, these path can result in qualitatively different kinds of minima. We can *not* draw any general conclusions about what kind of minima each algorithm finds based on this toy example, however we make two observations. First, as showed on the KL-surface plot for [3] the step taken by natural gradient can be smaller than gradient descent (i.e. the KL curvature is high) even though the error surface curvature is not high (i.e. Newton's method step is larger than gradient descent step). Secondly, the direction chosen by natural gradient can be quite different from that of gradient descent (see for example in [3] and [4]), which can result in finding a different local minima than gradient descent (for e.g. when the model starts at [3]).

4.1 A comparison between Amari's and Le Roux's natural gradient

In Le Roux *et al.* (2007) a different approach is taken to derive natural gradient. Specifically one assumes that the gradients computed over different minibatches are distributed according to a Gaussian centered around the true gradient with some covariance matrix $\mathbf{C}$. By using the uncertainty provided by $\mathbf{C}$ we can correct the step that we are taking to maximize the probability of a downward move in generalization error (expected negative log-likelihood), resulting in a formula similar to that

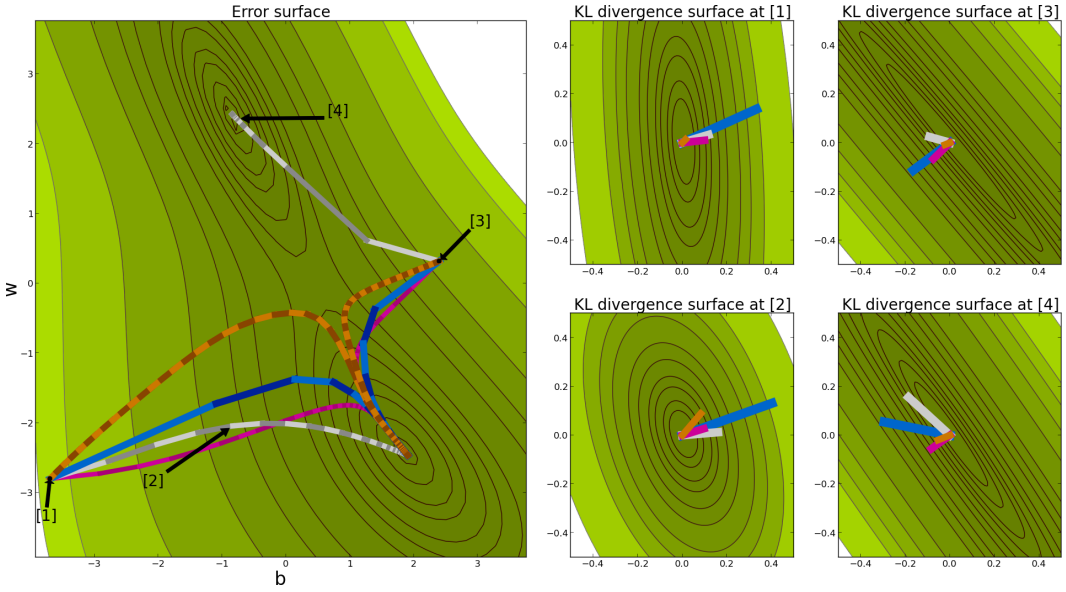

Figure 1: Path taken by four different learning algorithms towards a local minima. Newton method (blue), natural gradient (gray), Le Roux's natural gradient (orange) and gradient descent (purple). See text for details.

of natural gradient. If $\mathbf{g} = \frac{\partial \mathcal{L}}{\partial \theta}$ is the gradient, then Le Roux *et al.* (2007) proposes following the direction $\tilde{g} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\mathbf{C}^{-1}$ where $\mathbf{C}$ is:

$$\mathbf{C} = \frac{1}{n}\sum_i (\mathbf{g} - \langle \mathbf{g} \rangle)^T (\mathbf{g} - \langle \mathbf{g} \rangle) \tag{12}$$

While the probabilistic derivation requires the use of the centered covariance, equation (12), in Le Roux *et al.* (2007) it is argued that using the uncentered covariance $\mathbf{U}$ is equivalent up to a constant resulting in a simplified formula which is sometimes confused with the metric derived by Amari.

$$\mathbf{U} \quad = \quad \frac{1}{n}\sum_i \mathbf{g}^T\mathbf{g} \approx \mathbb{E}_{(\mathbf{x},\mathbf{t})\sim q}\left[\left(\frac{\partial \log p(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)^T \left(\frac{\partial \log p(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)\right] \tag{13}$$

The misunderstanding comes from the fact that the equation has the form of an expectation, though the expectation is over the empirical distribution $q(\mathbf{x}, \mathbf{t})$. It is therefore not clear if $\mathbf{U}$ tells us how p_θ would change, whereas it is clear that $\mathbf{G}$ does. The two methods are just different, and one can not straightforwardly borrow the interpretation of one for the other. However, we believe that there is an argument strongly suggesting that the protection against drops in generalization error afforded by Le Roux's $\mathbf{U}$ is also a property shared by the natural gradient's $\mathbf{G}$.

If $KL(p \parallel q)$ is small, than $\mathbf{U}$ can be seen as an approximation to $\mathbf{G}$. Specifically we approximate the second expectation from equation (8), i.e. the expectation over $\mathbf{t} \sim p_\theta(\mathbf{t}|\mathbf{x})$, by a single point, the corresponding $\mathbf{t}^{(i)}$. This approximation makes sense when $\mathbf{t}^{(i)}$ is a highly probable sample under p which happens when we converge. Note that at convergence, $\mathbf{U}, \mathbf{G}$ and the Hessian are very similar, hence both versions of natural gradient and most second order methods would behave similarly.

An interesting question is if these different paths taken by each algorithm represent qualitatively different kinds of solutions. We will address this question indirectly by enumerating what implications each choice has.

The first observation has to do with numerical stability. One can express $\mathbf{G}$ as a sum of $n \times o$ outer products (where n is the size of the minibatch over which we estimate the matrix and o is the number

of output units) while $\mathbf{U}$ is a sum of only n outer products. Since the number of terms in these sums provides an upper bound on the rank of each matrix, it follows that one could expect that $\mathbf{U}$ will be lower rank than $\mathbf{G}$ for the same size of the minibatch n. This is also pointed out by Schraudolph (2002) to motivate the extended Gauss-Newton matrix as middle ground between natural gradient and the true Hessian[3]

A second difference regards plateaus of the error surface. Given the formulation of our error function in equation (6) (which sums the log of $p_\theta(t|x)$ for specific values of t and x), flat regions of the objective function are intrinsically flat regions of the functional manifold[4]. Moving at constant speed in the functional space means we should not get stalled near such plateaus. In Park *et al.* (2000) such plateaus are found near singularities of the functional manifold, providing a nice framework to study them (as is done for example in Rattray *et al.* (1998) where they hypothesize that such singularities behave like repellors for the dynamics of natural gradient descent). An argument can also be made in favour of $\mathbf{U}$ at plateaus. If a plateau at θ exists for most possible inputs $\mathbf{x}$, than the covariance matrix will have a small norm (because the vectors in each outer product will be small in value). The inverse of $\mathbf{U}$ consequentially will be large, meaning that we will take a large step, possibly out of the plateau region. This suggest both methods should be able to escape from some plateaus, though the reasoning behind the functional manifold approach more clearly motivates this advantage.

Another observation that is usually made regarding the functional manifold interpretation is that it is parametrization-independent. That means that regardless of how we parametrize our model we should move at the same speed, property assured by the constraint on the KL-divergence between p_θ and $p_{\theta+\Delta\theta}$. In Sohl-Dickstein (2012), following this idea, a link is made between natural gradient and whitening in parameter space. This property does not transfer directly to the covariance matrix.

On the other hand Le Roux's method is designed to obtain better generalization errors by moving mostly in the directions agreed upon by the gradients on most examples. We will argue that *the functional manifold approach can also provide a similar property*.

One argument relies on large detrimental changes of the expected log-likelihood, which is what Le Roux's natural gradient step protects us from with higher probability. The metric of Amari's natural gradient measures the expected (over $\mathbf{x}$) KL-divergence curvature. We argue that if $\Delta\theta$ induces a large change in log-likelihood computed over $\mathbf{x} \in \mathbb{D}$, where $\mathbb{D}$ corresponds to some minibatch, then it produces a large change in p_θ (in the KL sense), i.e. it results in a high KL-curvature. Because we move at constant speed on the manifold, we slow down in these high KL-curvature regions, and hence we do not allow large detrimental changes to happen. This intuition becomes even more suitable when $\mathbb{D}$ is larger than the training set, for example by incorporating unlabeled data, and hence providing a more accurate measure of how p_θ changes (in the KL sense). This increase in accuracy should allow *for better predictions of large changes in the generalization error as opposed to only the training error*.

A second argument comes from looking at the Fisher Information matrix which has the form of an uncentered weighted covariance matrix of gradients, $\mathbb{E}_{\mathbf{x}}\left[\sum_{\mathbf{t}} p_\theta(\mathbf{t}|\mathbf{x}) \left(\frac{\partial \log p_\theta(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)^T \left(\frac{\partial \log p_\theta(\mathbf{t}|\mathbf{x})}{\partial \theta}\right)\right]$. Note that these are *not the gradients* $\frac{\partial \mathcal{L}}{\partial \theta}$ *that we follow towards a local minima*. By using this matrix natural gradient moves in the expected direction of low variance for p_θ. As the cost $\mathcal{L}$ just evaluates p_θ at certain points $\mathbf{t}^{(i)}$ for a given $\mathbf{x}^{(i)}$, we argue that with high probability expected directions of low variance for p_θ correspond to directions of low variance for $\mathcal{L}$. Note that directions of high variance for $\mathcal{L}$ indicates direction in which p_θ changes quickly which should be reflected in large changes of the KL. Therefore, in the same sense as TONGA, natural gradient avoids directions of high variance that can lead to drops in generalization error.

[3]Note that Schraudolph (2002) assumes a form of natural gradient that uses $\mathbf{U}$ as a metric, similar to the Le Roux *et al.* (2007) proposed, an assumption made in Martens (2010) as well.

[4]One could argue that p might be such that $\mathcal{L}$ has a low curvature while the curvature of KL is much larger. This would happen for example if p is not very sensitive to θ for the values $x^{(i)}, t^{(i)}$ provided in the training set, but it is for other pairings of x and t. However we believe that in such a scenario is more useful to move slowly, as the other parings of x and t might be relevant for the generalization error

4.2 Natural gradient descent versus second order methods

Even though natural gradient is usually assumed to be a second order method it is more useful, and arguably more correct to think of it as a first order method. While it makes use of curvature, it is the curvature of the functional manifold and not that of the error function we are trying to minimize. The two quantities are different. For example the manifold curvature matrix is positive semi-definite by construction while for the Hessian we can have negative curvature.

To make this distinction clear we can try to see what information carries the metric that we invert (as it was done in Roux and Fitzgibbon (2010) for Newton's and Le Roux's methods).

The functional manifold metric can be written as either the expectation of the Hessian $\frac{\partial^2 p_\theta}{\partial \theta^2}$ or the expectation of the Fisher Information Matrix $\left[\left(\frac{\partial p_\theta}{\partial \theta}\right)^T \frac{\partial p_\theta}{\partial \theta}\right]$ (see (3) and (4)). The first form tells us the that the matrix measures how a change in θ affects the gradients $\frac{\partial p_\theta}{\partial \theta}$ of p_θ (as the Hessian would do for the error). The second form tells us how the change in the input affects the gradients $\frac{\partial p_\theta}{\partial \theta}$, as the covariance matrix would do for Le Roux's TONGA. However, while the matrix measures both the effects of a change in the input and θ it does so on the functional behaviour of p_θ who acts as a surrogate for the training error. As a consequence we need to look for density functions p_θ which are correlated with the training error, as we do in the examples discussed here.

Lastly, compared to second order methods, natural gradient lends itself very well to the online optimization regime. In principle, in order to apply natural gradient we need an estimate of the gradient, *which can be the stochastic gradient over a single sample* and some reliable measure of how our model, through p_θ, changes with θ (in the KL sense), which is given by the metric. For e.g. in case of probabilistic models like DBMs, the metric relies only on negative samples obtained from p_θ and does not depend on the empirical distribution q at al Desjardins *et al.* (2013), while for a second order method the Hessian would depend on q. For conditional distributions (as is the case for neural networks), one good choice is to compute the metric on a held out subset of input samples, offering this way an unbiased estimate of how $p(\mathbf{t}|\mathbf{x})$ changes with θ. This can easily be done in an online regime. Given that we do not even need to have targets for the data over which we compute the metric, as $\mathbf{G}$ integrates out the random variable t, we could even *use unlabeled data* to improve the accuracy as long as it comes from the same distribution q, which can not be done for second order methods.

5 Natural gradient robustness to overfitting

We explore the robustness hypothesis from section 4.1 empirically. The results of all experiments carried out are summarized in table 1 present in the Appendix. Firstly we consider the effects of *using extra unlabeled data to improve the accuracy of the metric*. A similar idea was proposed in Sun *et al.* (2009). The idea is that for $\mathbf{G}$ to do a good job in this robustness sense, it has to accurately predict the change in KL divergence in every direction. If $\mathbf{G}$ is estimated from too little data (e.g., a small labeled set) and that data happens to be the training set, then it might "overfit" and underestimate the effect of a change in some directions where the training data would tend to push us. To protect us against this, what we propose here is the use a *large unlabeled set* to obtain a more generalization-friendly metric $\mathbf{G}$.

Figure 2 describes the results on the Toronto Face Dataset (TFD), where using unlabeled data results in 83.04% accuracy vs 81.13% without. State of the art is 85%Rifai *et al.* (2012), though this result is obtained by a larger model that is pre-trained. Hyper-parameters were validated using a grid-search (more details in the Appendix).

As you can see from the plot, it suggests that using unlabeled data helps to obtain better testing error, as predicted by our argument in Sec. 4.1. This comes at a price. Convergence (on the training error) is slower than when we use the same training batch.

Additionally we explore the effect of using different batches of training data to compute the metric. The full results as well as experimental setup are provided in the Appendix. It shows that, as most second order methods, natural gradient has a tendency to overfit the current minibatch if both the metric and the gradient are computed on it. However, as suggested in Vinyals and Povey (2012) using different minibatches for the metric helps as we tend not to ignore directions relevant for other minibatches.

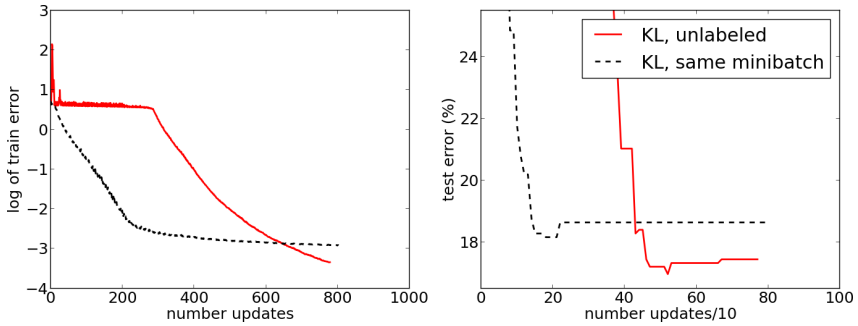

Figure 2: (left) train error (cross entropy over the entire training set) on a log scale and (right) test error (percentage of misclassified examples) as a function of number of updates for the Toronto Faces Dataset. 'kl, unlabeled' stands for the functional manifold version of natural gradient, where the metric is computed over unlabeled data. for 'KL, different training minibatch' we compute the metric on a different minibatch from the training set, while 'KL, same minibatch' we compute the metric over the same minibatch we computed the gradient hence matching the standard use of hessian-free. 'covariance' stands for tonga that uses the covariance matrix as a metric, while msgd is minibatch stochastic gradient descent. note that the x axis was interrupted, in order to improve the visibility of how the natural gradient methods behave.

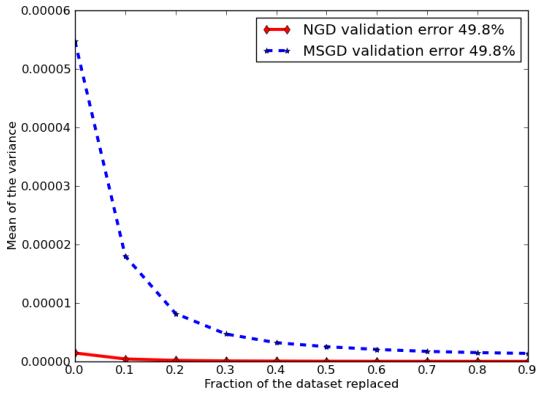

Figure 3: The plot describes how much the model is influenced by different parts of an online training set, for the two learning strategies compared (minibatch stochastic gradient descent and natural gradient descent). The x-axis indicates which part (1st 10th, 2nd 10th, etc.) of the first half of the data was randomly resampled, while the y-axis measures the resulting variance of the output due to the change in training data.

6 Natural gradient is robust to the order of the training set

We explore the regularization effects of natural gradient descent by looking at the variance of the trained model as a function of training samples that it sees. To achieve this we repeat the experiment described in Erhan *et al.* (2010) which looks at how resampling different fraction of the training set affects the variance of the model and focuses specifically to the relative higher variance of the early examples. Our intuition is that by forbidding large jumps in the KL divergence of p_θ and following the direction of low variance natural gradient will try to limit the amount of overfitting that occurs *at any stage of learning*.

We repeat the experiment from Erhan *et al.* (2010), using the NISTP dataset introduced in Bengio *et al.* (2011) (which is just the NIST dataset plus deformations) and use 32.7M samples of this data. We divide the first 16.3M data into 10 equal size segments. For each data point in the figure, we fix 9 of the 10 data segments, and over 5 different runs we replace the 10th with 5 different random sets

of samples. This is repeated for each of the 10 segments to produce the down curves. By looking at the variance of the model outputs on a held out dataset (of 100K samples) after the whole 32.7M online training samples, we visualize the influence of each of the 10 segments on the function learnt (i.e., at the end of online training). The curves can be seen in figure 3.

There are two observation to be made regarding this plot. Firstly, it seems that early examples have a relative larger effect on the behaviour of the function than latter ones (phenomena sometimes called early-overfitting). This happens for both methods, natural gradient and stochastic gradient descent. The second observation regards the overall variance of the learnt model.

Note that the variance at each point on the curve depends on the speed with which we move in functional space. For a fixed number of examples one can artificially tweak the curves for e.g. by decreasing the learning rate. With a smaller learning rate we move slower, and since the model, from a functional point of view, does not change by much, the variance is lower. In the limit, with a learning rate of 0, the model always stays the same. If we increase the number of steps we take (i.e. measure the variance after k times more samples) the curve recovers some of its shape.This is because we allow the model to move further away from the starting point.

In order to be fair to the two algorithms, we use the validation error as a measure of how much we moved in the functional space. This helps us to chose hyper-parameters such that after 32.7M samples both methods achieve the same validation error of 49.8% (see Appendix for hyper-parameters).

The results are consistent with our hypothesis that natural gradient avoids making large steps in function space during training, staying on the path that induces least variance. Such large steps may be present with SGD, possibly yielding the model to overfit (e.g. getting forced into some quadrant of parameter space based only on a few examples) resulting in different models at the end. By reducing the variance overall the natural gradient becomes more invariant to the order in which examples are presented. Note that the relative variance of early examples to the last re-sampled fraction is about the same for both natural gradient and stochastic gradient descent. However, the amount of variance induced in the learnt model by the early examples for natural gradient is on the same magnitude as the variance induce by the last fraction of examples for MSGD (i.e. in a global sense natural gradient is less sensitive the order of samples it sees).

7 The relationship between Hessian-Free and natural gradient

Hessian-Free as well as Krylov Subspace Descent rely on the extended Gauss-Newton approximation of the Hessian, $\mathbf{G_N}$ instead of the actual Hessian (see Schraudolph (2002)).

$$\mathbf{G_N} = \frac{1}{n}\sum_i \left[\left(\frac{\partial \mathbf{r}}{\partial \theta}\right)^T \frac{\partial^2 \log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)})}{\partial \mathbf{r}^2}\left(\frac{\partial \mathbf{r}}{\partial \theta}\right)\right] = \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_\mathbf{r}^T\left(\mathbb{E}_{\mathbf{t}\sim\tilde{q}(\mathbf{t}|\mathbf{x})}\left[\mathbf{H}_{\mathcal{L}\circ\mathbf{r}}\right]\right)\mathbf{J}_\mathbf{r}\right] \quad (14)$$

The reason is not computational, as computing both can be done equally fast, but rather better behaviour during learning. This is usually assumed to be caused by the fact that the Gauss-Newton is positive semi-definite by construction, so one needs not worry about negative curvature issues.

In this section we show that in fact the extended Gauss-Newton approximation matches perfectly the natural gradient metric, and hence by choosing this specific approximation, one can view both algorithms as being implementations of natural gradient rather than typical second order methods.

The last step of equation (14) is obtained by using the normal assumption that $(\mathbf{x}^{(i)}, \mathbf{t}^{(i)})$ are i.i.d samples. We will consider the three activation functions and corresponding errors for which the extended Gauss-Newton is defined and show it matches perfectly the natural gradient metric for the same activation.

For the linear output units with square errors we can derive the matrix $\mathbf{H}_{\mathcal{L}\circ\mathbf{r}}$ as follows:

$$\begin{aligned}\mathbf{H}_{\mathcal{L}\circ\mathbf{r}_{ij,i\neq j}} &= \frac{\partial^2 \sum_k (\mathbf{r}_k - \mathbf{t}_k)^2}{\partial \mathbf{r}_i \partial \mathbf{r}_j} = \frac{\partial 2(\mathbf{r}_i - \mathbf{t}_i)}{\partial \mathbf{r}_j} = 0 \\ \mathbf{H}_{\mathcal{L}\circ\mathbf{r}_{ii}} &= \frac{\partial^2 \sum_k (\mathbf{r}_k - \mathbf{t}_k)^2}{\partial \mathbf{r}_i \partial \mathbf{r}_i} = \frac{\partial 2(\mathbf{r}_i - \mathbf{t}_i)}{\partial \mathbf{r}_i} = 2\end{aligned} \quad (15)$$

$$\mathbf{G_N} = \frac{1}{n} \sum_{\mathbf{x}^{(i)}, \mathbf{t}^{(i)}} \mathbf{J_r}^T \mathbf{H}_{\mathcal{L} \circ \mathbf{r}} \mathbf{J_r} = \frac{1}{n} \sum_{\mathbf{x}^{(i)}, \mathbf{t}^{(i)}} \mathbf{J_y}^T \mathbf{H}_{\mathcal{L} \circ \mathbf{y}} \mathbf{J_y} = \frac{1}{n} \sum_{\mathbf{x}^{(i)}} \mathbf{J_y}^T (2\mathbf{I}) \mathbf{J_y} = 2\mathbb{E}_{\mathbf{x} \in q(\mathbf{x})} \left[\mathbf{J_y}^T \mathbf{J_y} \right] \tag{16}$$

The result is summarized in equation 16, where we make use of the fact that $\mathbf{r} = \mathbf{y}$. It matches the corresponding natural gradient metric, equation (24) from section 3, up to a constant.

In the case of sigmoid units with cross-entropy objective (σ is the sigmoid function), $\mathbf{H}_{\mathcal{L} \circ \mathbf{r}}$ is

$$\begin{array}{rcl} \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ij, i \neq j}} & = & \frac{\partial^2 \sum_k (-t_k \log(\sigma(r_k)) - (1-t_k) \log(1-\sigma(r_k)))}{\partial r_i \partial r_j} \\ & = & \frac{\partial \left(-t_i \frac{1}{\sigma(r_i)} \sigma(r_i)(1-\sigma(r_i)) + (1-t_i) \frac{1}{1-\sigma(r_i)} \sigma(r_i)(1-\sigma(r_i)) \right)}{\partial r_j} = \frac{\partial \sigma(r_i) - t_i}{\partial r_j} = 0 \\ \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ii}} & = & ... = \frac{\partial \sigma(r_i) - t_i}{\partial r_i} = \sigma(r_i)(1 - \sigma(r_i)) \end{array} \tag{17}$$

If we insert this back into the Gauss-Newton approximation of the Hessian and re-write the equation in terms of $\mathbf{J_y}$ instead of $\mathbf{J_r}$, we get, again, the corresponding natural gradient metric, equation (10).

$$\begin{array}{rcl} \mathbf{G_N} & = & \frac{1}{n} \sum_{\mathbf{x}^{(i)}, \mathbf{t}^{(i)}} \mathbf{J_r}^T \mathbf{H}_{\mathcal{L} \circ \mathbf{r}} \mathbf{J_r} = \frac{1}{n} \sum_{\mathbf{x}^{(i)}} \mathbf{J_r}^T \text{diag}\left(\mathbf{y}(1-\mathbf{y})\right) \text{diag}\left(\frac{1}{\mathbf{y}(1-\mathbf{y})} \right) \text{diag}\left(\mathbf{y}(1-\mathbf{y})\right) \mathbf{J_r} \\ & = & \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[\mathbf{J_y}^T \text{diag}\left(\frac{1}{\mathbf{y}(1-\mathbf{y})} \right) \mathbf{J_y} \right] \end{array} \tag{18}$$

The last matching activation and error function that we consider is the softmax with cross-entropy.

$$\begin{array}{rcl} \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ij, i \neq j}} & = & \frac{\partial^2 \sum_k (-t_k \log(\phi(r_k)))}{\partial r_i \partial r_j} = \frac{\partial \sum_k (t_k \phi(r_i)) - t_i}{\partial r_j} = -\phi(r_i)\phi(r_j) \\ \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ii}} & = & ... = \frac{\partial \phi(r_i) - t_i}{\partial r_i} = \phi(r_i) - \phi(r_i)\phi(r_i) \end{array} \tag{19}$$

Equation (20) starts from the natural gradient metric and singles out a matrix $\mathbf{M}$ in the formula such that the metric can be re-written as the product $\mathbf{J_r}^T \mathbf{M} \mathbf{J_r}$ (similar to the formula for the Gauss-Newton approximation). In (21) we show that indeed $\mathbf{M}$ equals $\mathbf{H}_{\mathcal{L} \circ \mathbf{r}}$ and hence the natural gradient metric is the same as the extended Gauss-Newton matrix for this case as well. Note that δ is the Kronecker delta, where $\delta_{ij, i \neq j} = 0$ and $\delta_{ii} = 1$.

$$\begin{array}{rcl} \mathbf{G} & = & \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[\sum_{k=1}^{o} \frac{1}{y_k} \left(\frac{\partial y_k}{\partial \theta} \right)^T \frac{\partial y_k}{\partial \theta} \right] = \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[\mathbf{J_r}^T \left(\sum_{k=1}^{o} \frac{1}{y_k} \left(\frac{\partial y_k}{\partial \mathbf{r}} \right)^T \left(\frac{\partial y_k}{\partial \mathbf{r}} \right) \right) \mathbf{J_r} \right] \\ & = & \frac{1}{N} \sum_{\mathbf{x}^{(i)}} \left(\mathbf{J_r}^T \mathbf{M} \mathbf{J_r} \right) \end{array} \tag{20}$$

$$\begin{array}{rcl} \mathbf{M}_{ij, i \neq j} & = & \sum_{k=1}^{o} \frac{1}{y_k} \frac{\partial y_k}{\partial r_i} \frac{\partial y_k}{\partial r_j} = \sum_{k=1}^{o} (\delta_{ki} - y_i) y_k (\delta_{kj} - y_j) = y_i y_j - y_i y_j - y_i y_j = -\phi(r_i)\phi(r_j) \\ \mathbf{M}_{ii} & = & \sum_{k=1}^{o} \frac{1}{y_k} \frac{\partial y_k}{\partial y_i} \frac{\partial y_k}{\partial r_j} = y_i^2 \left(\sum_{k=1}^{o} y_k \right) + y_i - 2y_i^2 = \phi(r_i) - \phi(r_i)\phi(r_i) \end{array} \tag{21}$$

8 Conclusion

In this paper we re-derive natural gradient, by imposing that at each step we follow the direction that minimizes the error function while resulting in a constant change in the KL-divergence of the probability density function that represents the model. This approach minimizes the amount of differential geometry needed, making the algorithm more accessible.

We show that natural gradient, as proposed by Amari, is *not the same* as the algorithm proposed by Le Roux et al, even though it has the same name. We highlight a few differences of each algorithm and hypothesis that Amari's natural gradient should exhibit the same robustness against overfitting that Le Roux's algorithm has, but for different reasons.

We explore empirically this robustness hypothesis, by proving better test errors when *unlabeled data* is used to improve the accuracy of the metric. We also show that natural gradient may reduce the worrisome early specialization effect previously observed with online stochastic gradient descent applied to deep neural nets, and reducing the variance of the resulting learnt function (with respect to the sampled training data).

By computing the specific metrics needed for standard output activation functions we showed that the extended Gauss-Newton approximation of the Hessian coincides with the natural gradient metric (provided that the metric is estimated over the same batch of data as the gradient). Given this identity one can re-interpret the recently proposed Hessian-Free and Krylov Subspace Descent as natural gradient.

Finally we point out a few differences between typical second order methods and natural gradient. The latter seems more suitable for online or probabilistic models, and relies on a surrogate probability density function p_θ in place of the error function in case of deterministic models.

Acknowledgements

We would like to thank Guillaume Desjardens, Aaron Courville, Li Yao, David Warde-Farley and Ian Goodfellow for the interesting discussion on the topic, or for any help provided during the development of this work. Reviewers at ICLR were particularly helpful, and we want to thank them, especially one of the reviewers that suggested several links with work from the reinforcement learning community. Also special thanks goes to the Theano development team as well (particularly to Frederic Bastien, Pascal Lamblin and James Bergstra) for their help.

We acknowledge NSERC, FQRNT, CIFAR, RQCHP and Compute Canada for the resources they provided.

References

Amari, S. (1985). Differential geometrical methods in statistics. *Lecture notes in statistics*, **28**.

Amari, S. (1997). Neural learning in structured parameter spaces - natural Riemannian gradient. In *In Advances in Neural Information Processing Systems*, pages 127–133. MIT Press.

Amari, S., Kurata, K., and Nagaoka, H. (1992). Information geometry of Boltzmann machines. *IEEE Trans. on Neural Networks*, **3**, 260–271.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Comput.*, **10**(2), 251–276.

Arnold, L., Auger, A., Hansen, N., and Olivier, Y. (2011). Information-geometric optimization algorithms: A unifying picture via invariance principles. *CoRR*, **abs/1106.3708**.

Bengio, Y., Bastien, F., Bergeron, A., Boulanger-Lewandowski, N., Breuel, T., Chherawala, Y., Cisse, M., Côté, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Pannetier Lebeuf, S., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In *JMLR W&CP: Proc. AISTATS'2011*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Chapelle, O. and Erhan, D. (2011). Improved Preconditioner for Hessian Free Optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Choi, S.-C. T., Paige, C. C., and Saunders, M. A. (2011). MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. **33**(4), 1810–1836.

Desjardins, G., Pascanu, R., Courville, A., and Bengio, Y. (2013). Metric-free natural gradient for joint-training of boltzmann machines. *CoRR*, **abs/1301.3545**.

Erhan, D., Courville, A., Bengio, Y., and Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *JMLR W&CP: Proc. AISTATS'2010*, volume 9, pages 201–208.

Kakade, S. (2001). A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 1531–1538. MIT Press.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. (2007). Topmoumoute online natural gradient algorithm. Technical Report 1299, Département d'informatique et recherche opérationnelle, Université de Montréal.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. In *NIPS'07*.

Le Roux, N., Bengio, Y., and Fitzgibbon, A. (2011). Improving first and second-order methods by modeling uncertainty. In *Optimization for Machine Learning*. MIT Press.

Martens, J. (2010). Deep learning via hessian-free optimization. In *ICML*, pages 735–742.

Nocedal, J. and Wright, S. J. (2000). *Numerical Optimization*. Springer.

Park, H., Amari, S.-I., and Fukumizu, K. (2000). Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, **13**(7), 755 – 764.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *CoRR*, **abs/1211.5063**.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, **6**, 147–160.

Peters, J. and Schaal, S. (2008). Natural actor-critic. (7-9), 1180–1190.

Rattray, M., Saad, D., and Amari, S. I. (1998). Natural Gradient Descent for On-Line Learning. *Physical Review Letters*, **81**(24), 5461–5464.

Rifai, S., Bengio, Y., Courville, A., Vincent, P., and Mirza, M. (2012). Disentangling factors of variation for facial expression recognition. In *Proceedings of the European Conference on Computer Vision (ECCV 6)*, pages 808–822.

Roux, N. L. and Fitzgibbon, A. W. (2010). A fast natural newton method. In J. Fürnkranz and T. Joachims, editors, *ICML*, pages 623–630. Omnipress.

Schaul, T. (2012). Natural evolution strategies converge on sphere functions. In *Genetic and Evolutionary Computation Conference (GECCO)*.

Schraudolph, N. N. (2001). Fast curvature matrix-vector products. In *ICANN*, pages 19–26.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, **14**(7), 1723–1738.

Sohl-Dickstein, J. (2012). The natural gradient by analogy to signal whitening, and recipes and tricks for its use. *CoRR*, **abs/1205.1828**.

Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Stochastic search using the natural gradient. In *ICML*, page 146.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *ICML*, pages 1017–1024.

Vinyals, O. and Povey, D. (2012). Krylov Subspace Descent for Deep Learning. In *AISTATS*.

Appendix

8.1 Expected Hessian to Fisher Information Matrix

The Fisher Information Matrix form can be obtain from the expected value of the Hessian :

$$\begin{aligned}
\mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta}{\partial \theta}\right] &= \mathbb{E}_{\mathbf{z}}\left[-\frac{\partial \frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}}{\partial \theta}\right] = \mathbb{E}_{\mathbf{z}}\left[-\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta}{\partial \theta^2} + \left(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}\right)^T\left(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}\right)\right] \\
&= -\frac{\partial^2}{\partial \theta^2}\left(\sum_{\mathbf{z}} p_\theta(\mathbf{z})\right) + \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right] \\
&= \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right] \qquad (22)
\end{aligned}$$

8.2 Derivation of the natural gradient metrics

8.2.1 Linear activation function

In the case of linear outputs we assume that each entry of the vector t, t_i comes from a Gaussian distribution centered around $\mathbf{y}_i(\mathbf{x})$ with some standard deviation β. From this it follows that:

$$p_\theta(\mathbf{t}|\mathbf{x}) = \prod_{i=1}^{o} \mathcal{N}(t_i|\mathbf{y}(\mathbf{x},\theta)_i, \beta^2) \qquad (23)$$

$$
\begin{aligned}
\mathbf{G} &= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[\sum_{i=1}^{o}\left(\frac{\partial \log_\theta p(t_i|\mathbf{y}(\mathbf{x})_i}{\partial\theta}\right)^T\left(\frac{\partial \log p_\theta(t_i|\mathbf{y}(\mathbf{x})_i}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[\left(\frac{\partial(t_i-y_i)^2}{\partial\theta}\right)^T\left(\frac{\partial(t_i-y_i)^2}{\partial\theta}\right)\right]\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^{\mathbf{I}})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_{\mathbf{y}}^T\mathbf{J}_{\mathbf{y}}\right]
\end{aligned}
\tag{24}
$$

8.2.2 Sigmoid activation function

In the case of the sigmoid units, i,e, $\mathbf{y} = \text{sigmoid}(\mathbf{r})$, we assume a binomial distribution which gives us:

$$
p(\mathbf{t}|\mathbf{x}) = \prod_i \mathbf{y}_i^{t_i}(1-\mathbf{y}_i)^{1-t_i} \tag{25}
$$

$\log p$ gives us the usual cross-entropy error used with sigmoid units. We can compute the Fisher information matrix as follows:

$$
\begin{aligned}
\mathbf{G} &= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbb{E}_{\mathbf{t}\sim p(\mathbf{t}|\mathbf{x})}\left[\sum_{i=1}^{o}\frac{(t_i-y_i)^2}{y_i^2(1-y_i)^2}\left(\frac{\partial y_i}{\partial\theta}\right)^T\frac{\partial y_i}{\partial\theta}\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\frac{1}{y_i(1-y_i)}\left(\frac{\partial y_i}{\partial\theta}\right)^T\frac{\partial y_i}{\partial\theta}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_{\mathbf{y}}^T\text{diag}(\tfrac{1}{\mathbf{y}(1-\mathbf{y})})\mathbf{J}_{\mathbf{y}}\right]
\end{aligned}
\tag{26}
$$

8.2.3 Softmax activation function

For the softmax activation function, $\mathbf{y} = \text{softmax}(\mathbf{r})$, $p(\mathbf{t}|\mathbf{x})$ takes the form of a multinomial:

$$
p(\mathbf{t}|\mathbf{x}) = \prod_i^o \mathbf{y}_i^{t_i} \tag{27}
$$

$$
\mathbf{G} = \mathbf{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_i^o \frac{1}{\mathbf{y}_i}\left(\frac{\partial y_i}{\partial\theta}\right)^T\frac{\partial y_i}{\partial\theta}\right] \tag{28}
$$

8.3 Implementation Details

We have implemented natural gradient descent using a truncated Newton approach similar to the pipeline proposed by Pearlmutter (1994) and used by Martens (2010). In order to better deal with singular and ill-conditioned matrices we use the MinRes-QLP algorithm (Choi *et al.*, 2011) instead of linear conjugate gradient. Both Minres-QLP as well as linear conjugate gradient can be found implemented in Theano at https://github.com/pascanur/theano_optimize. We used the Theano library (Bergstra *et al.*, 2010) which allows for a flexible implementation of the pipeline, that can automatically generate the computational graph of the metric times some vector for different models:

```
import theano.tensor as TT
# `params` is the list of Theano variables containing the parameters
# `vs` is the list of Theano variable representing the vector `v`
# with whom we want to multiply the metric
# `Gvs` is the list of Theano expressions representing the product
# between the metric and `vs`

# `out_smx` is the output of the model with softmax units
Gvs = TT.Lop(out_smx,params,
             TT.Rop(out_smx,params,vs)/(out_smx*out_smx.shape[0]))

# `out_sig` is the output of the model with sigmoid units
Gvs = TT.Lop(out_sig,params,
             TT.Rop(out_sig,params,vs)/(out_sig*
                                        (1-out_sig)*
                                        out_sig.shape[0]))

# `out` is the output of the model with linear units
Gvs = TT.Lop(out,params,TT.Rop(out,params,vs)/out.shape[0])
```

The full pseudo-code of the algorithm (which is very similar to the one for Hessian-Free) is given below. The full Theano implementation can be retrieved from https://github.com/pascanur/natgrad.

Algorithm 1 Pseudocode for natural gradient algorithm

```
# 'gfn' is a function that computes the metric times some vector
gfn ← (lambda v → Gv)
while not early_stopping_condition do
    g ← ∂L/∂θ
    # linear_cg solves the linear system Gx = ∂L/∂θ
    ng← linear_cg(gfn, g, max_iters = 20, rtol=1e-4)
    # γ is the learning rate
    θ ← θ − γ ng
end while
```

Even though we are ensured that $\mathbf{G}$ is positive semi-definite by construction, and MinRes-QLP is able to find a suitable solutions in case of singular matrices, we still use a damping strategy for two reasons. The first one is that we want to take in consideration the inaccuracy of the metric (which is approximated only over a small minibatch). The second reason is that natural gradient makes sense only in the vicinity of θ as it is obtained by using a Taylor series approximation, hence (as for ordinary second order methods) it is appropriate to enforce a trust region for the gradient. See Schaul (2012), where the convergence properties of natural gradient (in a specific case) are studied.

Following the functional manifold interpretation of the algorithm, we can recover the Levenberg-Marquardt heuristic used in Martens (2010) by considering a first order Taylor approximation, where for any function f,

$$f\left(\theta_t - \eta \mathbf{G}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T\right) \approx f(\theta_t) - \eta \frac{\partial f(\theta_t)}{\partial \theta_t} \mathbf{G}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T \tag{29}$$

This gives as the reduction ratio given by equation (30) which can be shown to behave identically with the one in Martens (2010).

$$\rho = \frac{f\left(\theta_t - \eta \mathbf{G}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T\right) - f(\theta_t)}{-\eta \frac{\partial f(\theta_t)}{\partial \theta_t} \mathbf{G}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T} \tag{30}$$

8.4 Additional experimental results

For the one hidden unit auto-encoder we selected hyper-parameters such to ensure stability of training, while converging as fast as possible to a minima. We compute the inverse of the metric or Hessian exactly (as it is just a 2 by 2 matrix). The learning rate for SGD is set to .1, for Amari's natural gradient .5 and for the covariance of gradience 1. (Newton's method usually does not use a learning rate). We damped the Hessian and the covariance of gradients by adding $\mathbf{I}$ and Amari's metric using $0.01 \cdot \mathbf{I}$.

8.5 Restricted MNIST experiment

For the restricted MNIST, we train a one hidden layer MLP of 1500 hidden units. The hyper-parameters where chosen based on a grid search over learning rate, damping factor and damping strategy. Note that beside using unlabeled data, the regularization effect of natural gradient is strongly connected to the damping factor which accounts for the uncertainty in the metric (in a similar way to how it does in the uncentered covariance version of natural gradient). The minibatch size was kept constant to 2500 samples for natural gradient methods and 250 for MSGD. We used a constant learning rate and used a budged of 2000 iterations for natural gradient and 40000 iterations for MSGD.

We used a learning rate of 1.0 for MSGD and 5.0 for the functional manifold NGD using unlabeled data or the covariance based natural gradient. For the functional manifold NGD using either the same training minibatch or a different batch from the training set for computing the metric we set the learning rate to 0.1. We use a Levenberg-Marquardt heuristic only when using unlabeled data, otherwise the damping factor was kept constant. Its initial value was 2.0 for when using unlabeled data, and 0.01 for every case except when using the covariance of the gradients as the metric, when is set to 0.1.

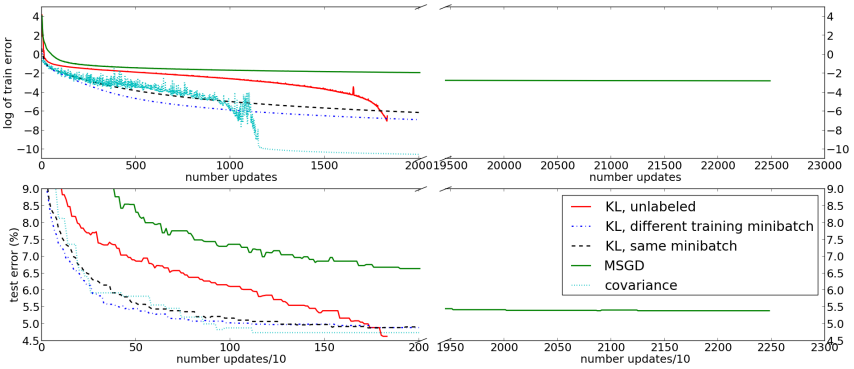

Figure 4: (left) train error (cross entropy over the entire training set) on a log scale in order to improve visibility and (right) test error (percentage of misclassified examples) as a function of number of updates for the restricted mnist dataset.

8.6 MNIST experiment

The model used has 3 layers, where the first two are convolutional layers both with filters of size 5x5. We used 32 filters on the first layer and 64 on the second. The last layer forms an MLP with 750 hidden units. We used minibatches of 10000 examples (for both the gradient and the metric), and a $\frac{1}{t}$ decaying learning rate strategy. The learning rate was kept constant for the first 200 updates and then it was computed based on the formula $\frac{l_0}{1+\frac{t-200}{20}}$, where t is the number of the current update. We used a budged ot 2000 update.

The learning rate was set to 0.5 for the functional manifold approach when using a different batch for computing the metric and 1.0 when using the same batch for computing the metric, or for using the covariance of gradients as metric. We use a Levenberg-Marquardt heuristic to adapt the damping

Table 1: Results on the three datasets considered (restricted MNIST, MNIST and TFD). Note that different models are used for different datasets. The training error is given as cross-entropy error, while the test error is percentage of miss-classified examples. The algorithms name are the same as in the legend of figure 2

DATA SET	DATA FOLD	MSGD	KL, UNLABELED	KL, DIFFERENT BATCH	KL, SAME BATCH	COVARIANCE
RESTRICTED MNIST	TRAIN	0.0523	0.0017	0.0012	0.0023	0.0006
	TEST	5.22%	4.63%	4.89%	4.91%	4.74%
MNIST	TRAIN			0.00010	0.0011	0.024
	TEST			0.78%	0.82%	1.07%
TFD	TRAIN		0.054	0.098		
	TEST		16.96%	18.87%		

factor which initially is 5.0 for the functional manifold approach, and a constant damping factor of 0.1 for using the covariance as metric. These values were validated by a grid search.

8.7 TFD experiment

The Toronto Face Dataset (TFD), has a large amount of unlabeled data of poorer quality than the training set. To ensure that the noise in the unlabeled data does not affect the metric, we compute the metric over the training batch plus unlabeled samples. We used a three hidden layer model, where the first layer is a convolutional layer of 300 filters of size 12x12. The second two layers from a 2 hidden layer MLP of 2048 and 1024 hidden units respectively.

For the TFD experiment we used the same decaying learning rate strategy introduced above, in subsection 8.6, where we computed gradients over the minibatch of 960 examples. When using the unlabeled data, we added 480 unlabeled examples to the 960 used to compute the gradient (therefore the metric was computed over 1440 examples) otherwise we used the same 960 examples for the metric. In both cases we used an initial damping factor of 8, and the Levenberg-Marquardt heuristic to adapt this damping value. Initial learning rate l_0 was set to 1 in both cases.

Note that we get only 83.04% accuracy on this dataset, when the state of the art is 85.0% Rifai *et al.* (2012), but our first layer is roughly 3 times smaller (300 filters versus 1024).

8.8 NISTP exepriment (robustness to the order of training samples)

The model we experimented with was an MLP of only 500 hidden units. We compute the gradients for both MSGD and natural gradient over minibatches of 512 examples. In case of natural gradient we compute the metric over the same input batch of 512 examples. Additionally we use a constant damping factor of 3 to account for the noise in the metric (and ill-conditioning since we only use batches of 512 samples). The learning rates were kept constant, and we use .2 for the natural gradient and .1 for MSGD.

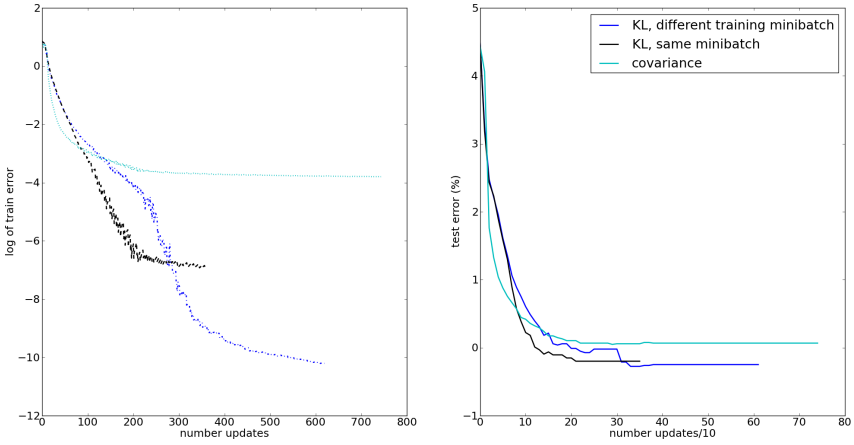

Figure 5: Train and test error (cross entropy) on a log scale as a function of number of updates for the MNIST dataset. The legend is similar to figure 2